Cloud Thinking

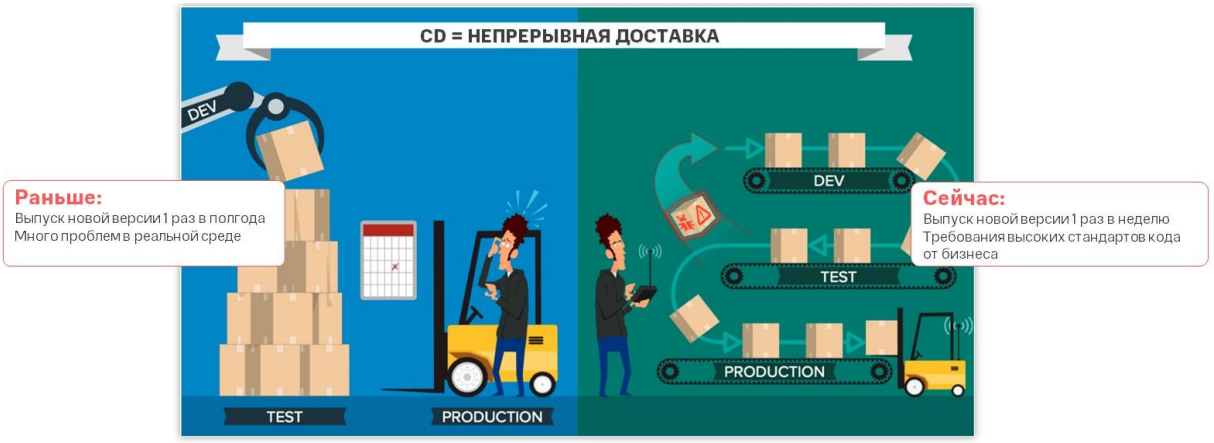
Модуль 2, урок 1

В этом уроке мы обсудим Kubernetes, контейнерные технологии и микросервисные решения. Прежде всего поговорим о том, почему они возникли на рынке и зачем нужны.

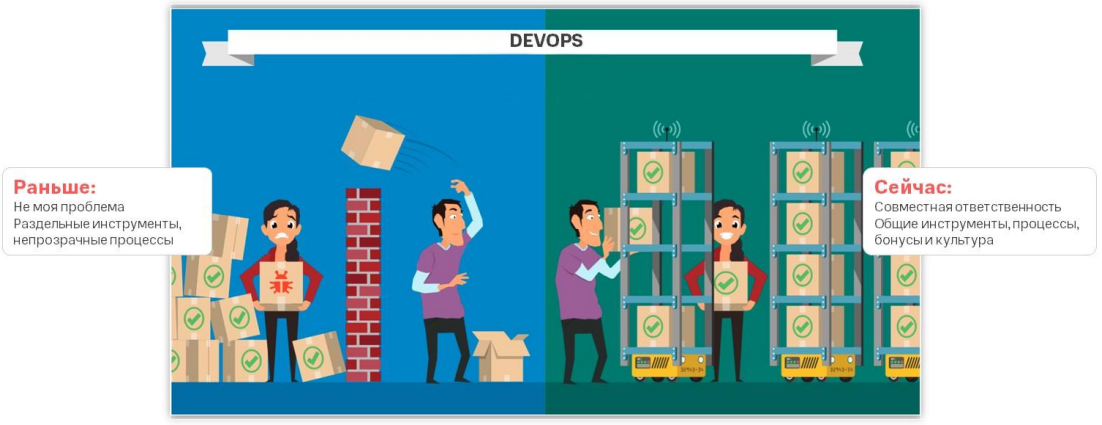
Традиционный подход vs DevOps

Мир не стоит на месте и динамично развивается, а вместе с ним развиваются цифровые технологии. Выигрывает гонку тот, кто выпустил на рынок новое приложение или внедрил функциональность быстрее, чем конкурент.

У этих приложений, как правило, есть большой и сложный бэкенд, который работает где-то в [облаке провайдера #CloudMTS]() или в вашем личном ЦОДе.

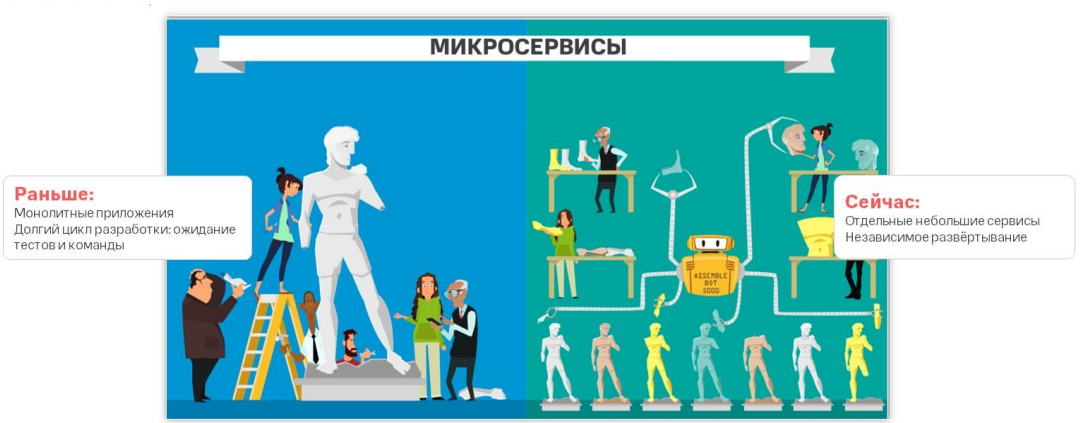


Разные команды внутри одной компании часто тянут одеяло на себя, не хотят делить ответственность и договариваться. Такие отношения еще работают, когда бизнес выпускает релизы продукта раз в полгода или год. Но что делать, если релизы нужны буквально каждую неделю? Это вынуждает команды садиться за один стол, договариваться и находить новые пути взаимодействия и совместной работы.

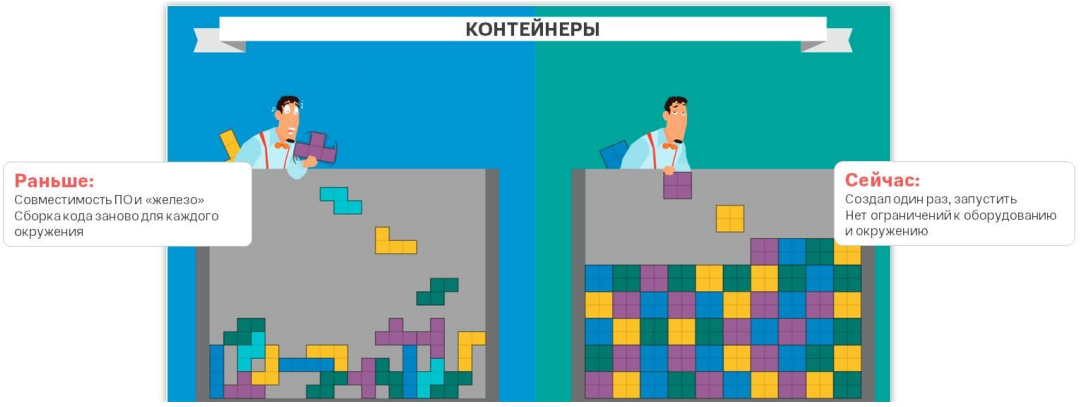


С другой стороны, стиль разработки тоже требует изменений. Традиционные монолитные приложения, хоть и прекрасно работают, достаточно сложны и медлительны. Их непросто запускать и практически невозможно масштабировать. Более того, изменение кодовой базы может повлечь за собой нежелательные артефакты и сбои в работе самого приложения.

Все это вынуждает искать новые подходы и рассматривать в качестве решения технологии микросервисов, которые позволяют разбить приложение на модули, работающие отдельно, но выполняющие некую единую логическую функцию.

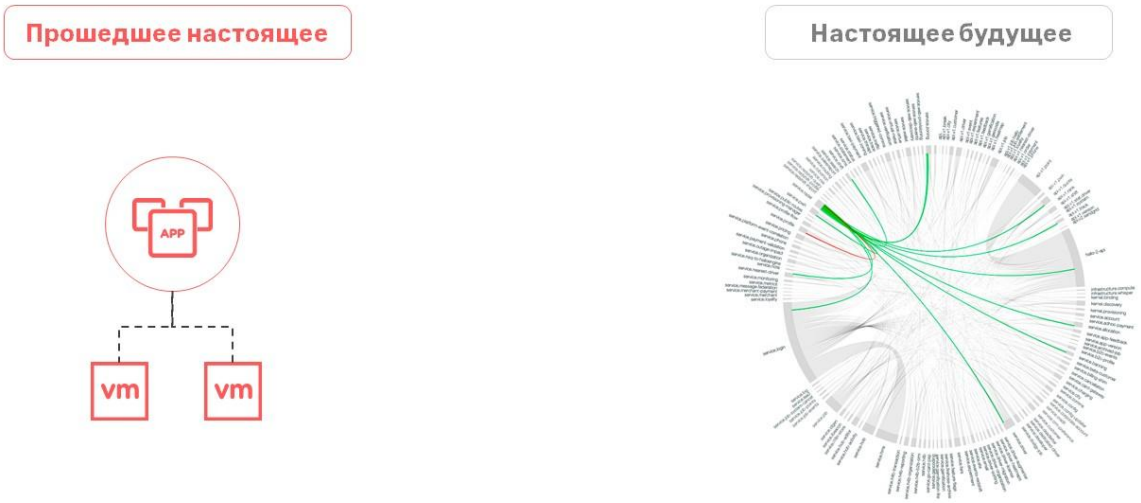


Общая тенденция к использованию облаков потребовала новых способов запуска приложений. Хорошо, когда разработчик может запустить приложение у себя на ноутбуке и оно прекрасно работает. Но после этого приложение требуется запустить, например, в локальной инфраструктуре или в инфраструктуре облачного провайдера.



Также не стоит забывать, что сами приложения стали сложнее. Раньше сложным приложением называлась сущность из нескольких виртуальных машин, которые отвечают за разный функционал. Сейчас сложные приложения — это сотни и даже тысячи различных модулей и подов, которые масштабируются, работают в связке «многие ко многим» и для запуска требуют довольно серьезной и детальной архитектурной проработки.

Приложения тогда и сейчас



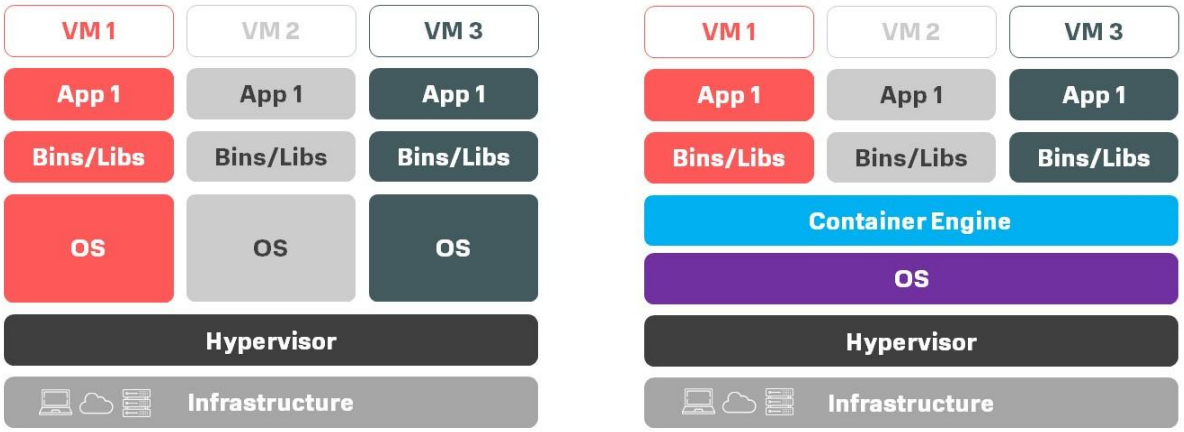
Итак, становится понятна общая тенденция к использованию микросервисов и контейнеров. Но как же они все-таки работают? Давайте попробуем разобраться.

Проще всего объяснять новое на контрасте с чем-то давно известным. В случае с контейнерными технологиями таким контрастом могут служить традиционные среды виртуализации и виртуальные машины. В традиционной архитектуре есть:

- сервер, на котором работает гипервизор;
- виртуальная машина, запущенная внутри гипервизора;
- операционная система внутри виртуальной машины;
- приложение, работающее поверх операционной системы.

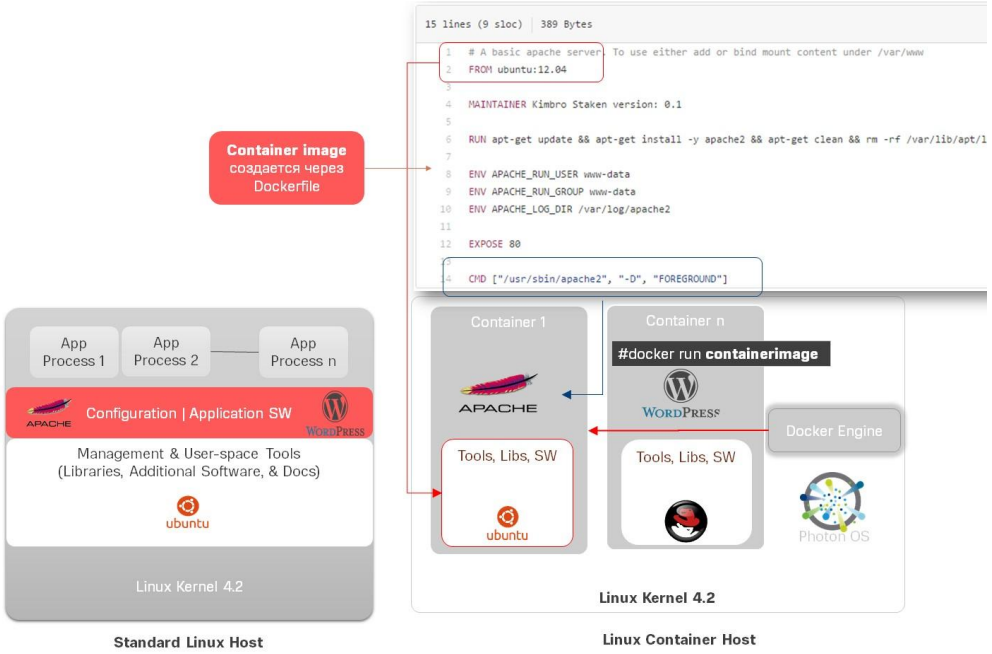
Контейнерные технологии дополняют это еще одним слоем абстракции. Внутри базовой операционной системы появляется контейнерный движок. Он позволяет запускать несколько различных приложений, фактически виртуализируя их. Благодаря этому подходу становится возможным запуск даже таких приложений, которые традиционно конфликтовали между собой и не могли работать внутри одной ОС.

VM vs Containers



Зачем нужны контейнеры

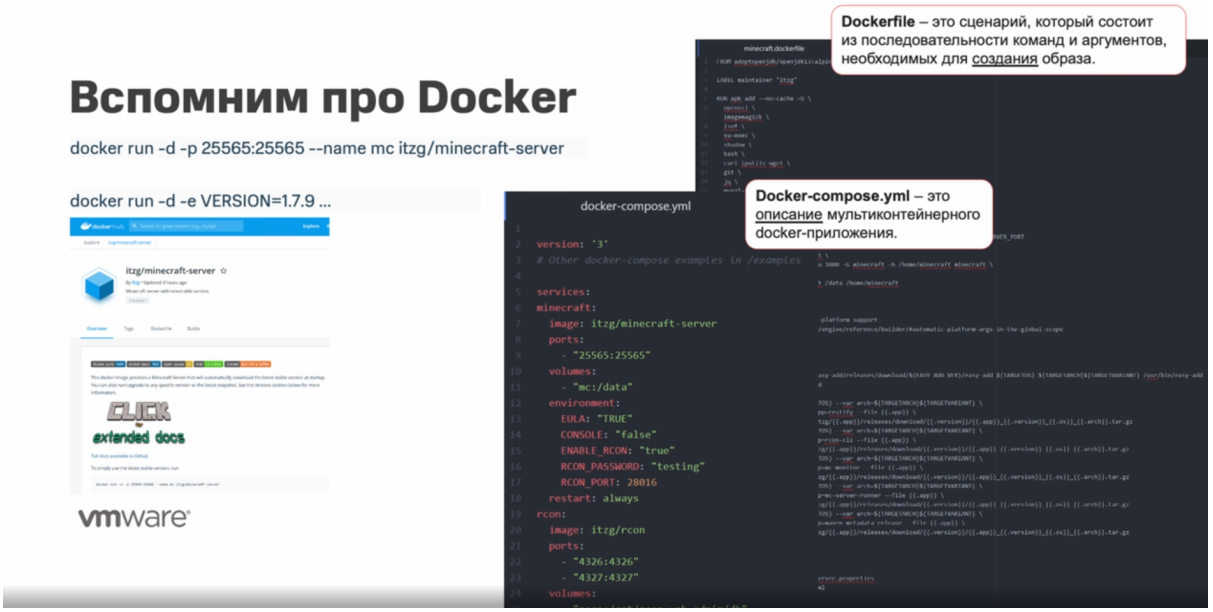
Ответ достаточно прост и понятен. Контейнеры нужны для того, чтобы ускорять и облегчать разработку и внедрение новых приложений в продуктив. Теперь разработчик может скомпилировать свой код, упаковать его в контейнер и запустить как у себя на ноутбуке, так и в продуктиве — в локальной инфраструктуре или в инфраструктуре облачного провайдера.



Вспомним про Docker

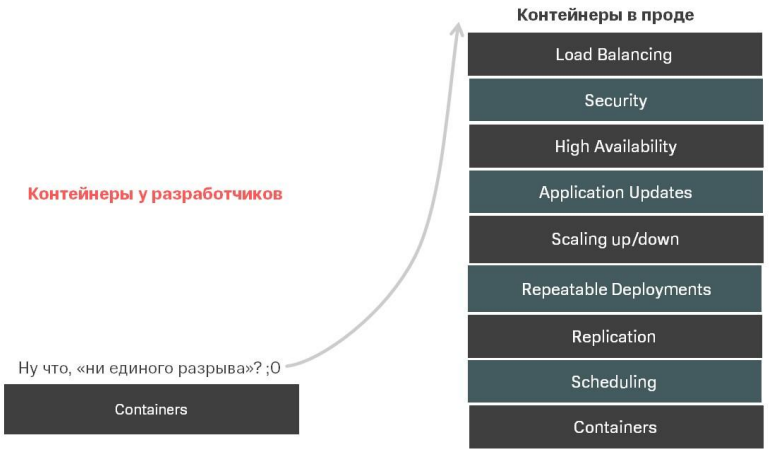
Docker — самый популярный на текущий момент контейнерный движок — позволяет разработчику получить две главные для него функции:

- Dockerfile позволяет описать командами действия, необходимые для того, чтобы скомпилировать и упаковать код в некий образ.
- Docker-compose.yml позволяет этот образ запустить с необходимыми конфигурациями, то есть получить некий готовый сервис.



Будем откровенны: все это прекрасно работает на бумаге и на ноутбуке разработчика. Но жизнь гораздо интереснее, и запуск этого приложения в продуктиве вызывает ряд новых и интересных задач.

Новые «вызовы» при использовании контейнеров



Что делать с масштабированием, отказоустойчивостью, безопасностью? Ведь не секрет, что 95% всех контейнеров, доступных в публичных реестрах, содержат различные уязвимости. Сразу возникает ряд вопросов, связанных с сетевой обвязкой, балансировкой нагрузки и обеспечением безопасности сетевых коммуникаций.

Эта прекрасная контейнерная история так и могла бы остаться просто историей, но тонущий контейнеровоз удалось спасти.

Что такое Kubernetes

Компания Google придумала систему для управления и тиражирования контейнерных приложений — тот самый Kubernetes.

Kubernetes - это open-source платформа для управления, автоматизации развертывания, масштабирования и управления контейнеризированными приложениями кластера рабочих узлов (worker nodes).

Ключевые возможности:

- Скорость и повторяемость развертывания
- Масштабирование «на лету»
- Простой выкат новых релизов
- Оптимизация затрат ресурсов

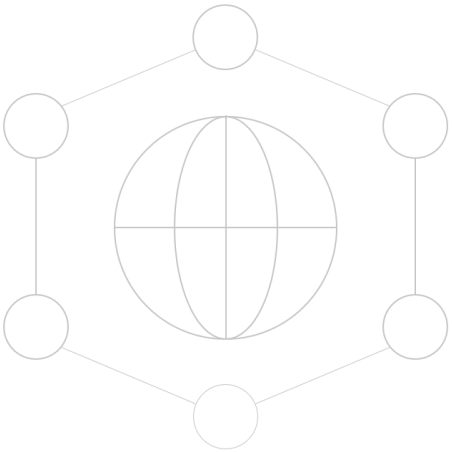
kubernetes

Сам по себе Kubernetes — прекрасная технология, но он не позволяет полноценно запуститься в продуктиве. К счастью, он изначально был разработан с учетом того, что в него будут интегрироваться дополнительные модули.

Kubernetes решает ряд проблем самих контейнеров, но при этом привносит много новых задач. Более того, сам он состоит из ряда сущностей, которые необходимо знать и понимать, как они работают, чтобы правильно оперировать этим оркестратором.

Сущности K8s

Namespace	• RBAC
Pod	• ServiceAccount
ReplicaSet	• User
Deployment	• Group
DeamonSet	• Role
Job / CronJob	• RoleBinding
ConfigMap	• ClusterRole
Secret	• ClusterRoleBinding

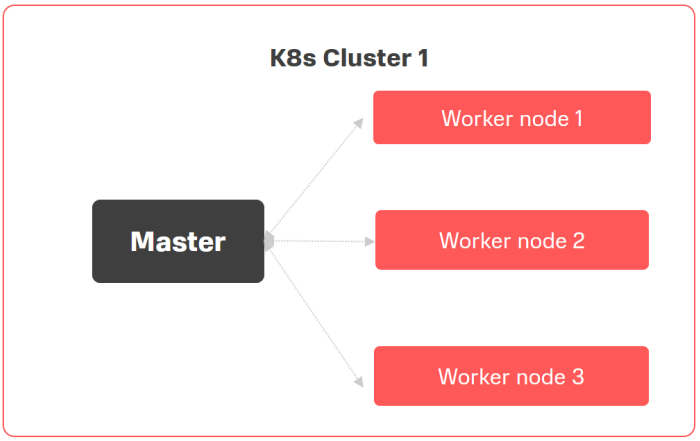


Обобщенно говоря, Kubernetes основан на технологиях кластеризации. Он состоит из двух основных сущностей — master-серверов и worker-серверов.

Master + worker nodes

Namespaces

Механизм разделения ресурсов в логические группы нагрузок

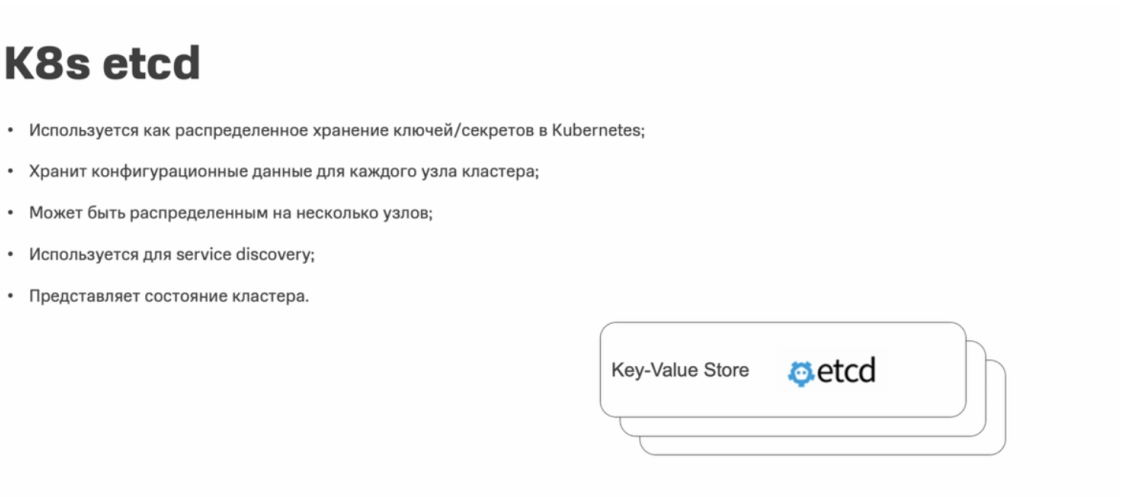


- Master-серверы управляют инфраструктурой.

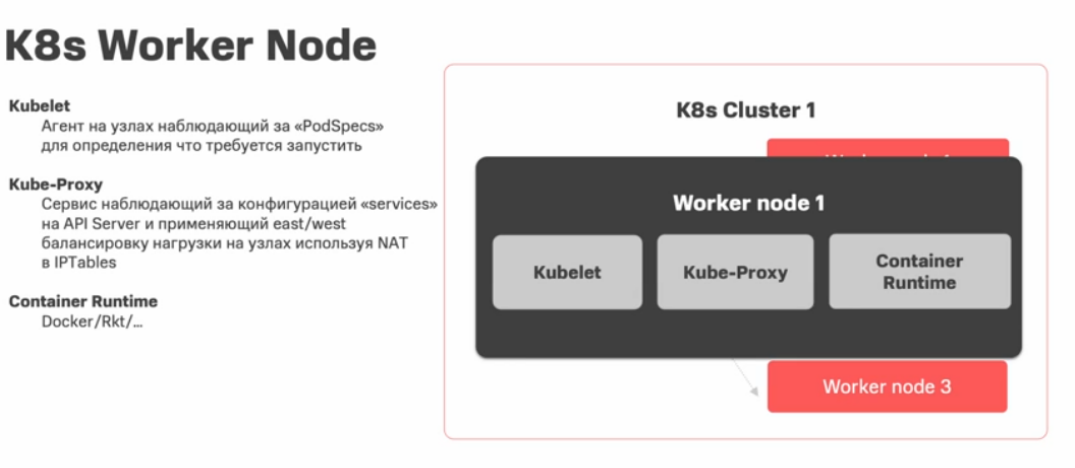
Они кластеризуются между собой и играют роль основной точки подключения для управления всей инфраструктурой.

Master-серверы хранят в себе распределенную базу данных Etcd с различными настройками и конфигурациями. Etcd используется для хранения сразу ряда сущностей, например секретов.

Секреты — это способы скрыть некую конфиденциальную информацию от посторонних глаз. В них часто хранятся данные о сертификатах, ключи для подключения по SSH или зашифрованные пароли, которые могут понадобиться подам для работы, но которые хотелось бы каким-то образом защитить от несанкционированного доступа.

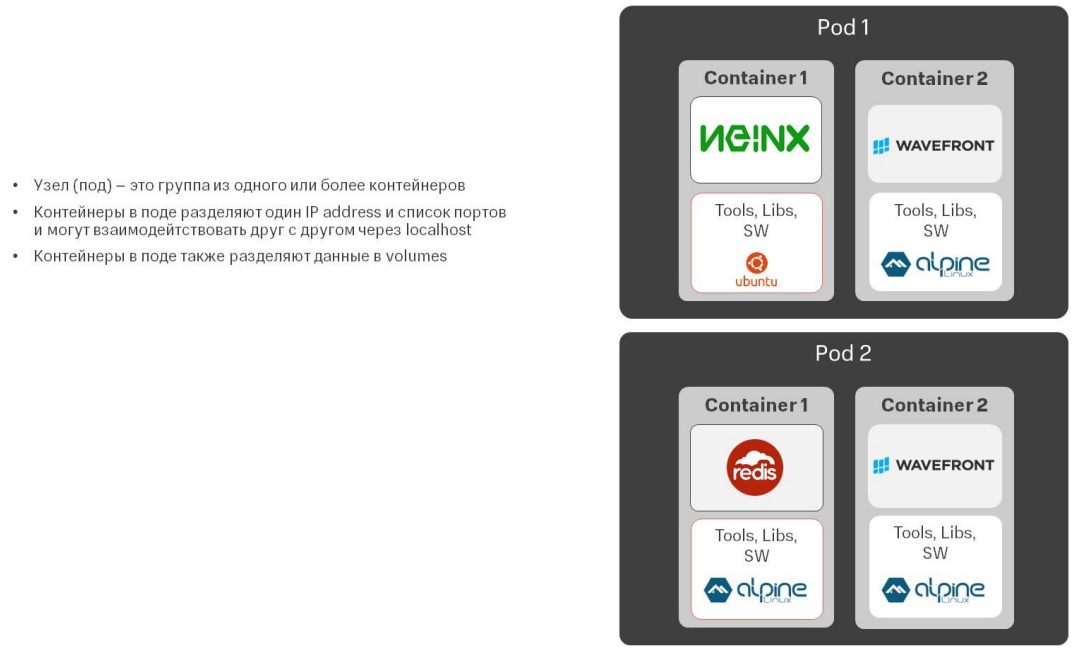


- На worker-серверах работают поды с приложениями.



Worker-серверы — это рабочие лошадки, на которых работают поды с приложениями. Внутри них тоже существует ряд сущностей. Например, Kubelet следит за правильной работой контейнеров, сверяя их с заранее созданной конфигурацией.
Kube-Proxy является некой точкой проксирования и распределения входящего трафика.
Внутри них существует разделение на пространство имен, Namespaces, которые позволяют логически разделить ресурсы между проектами и командами.

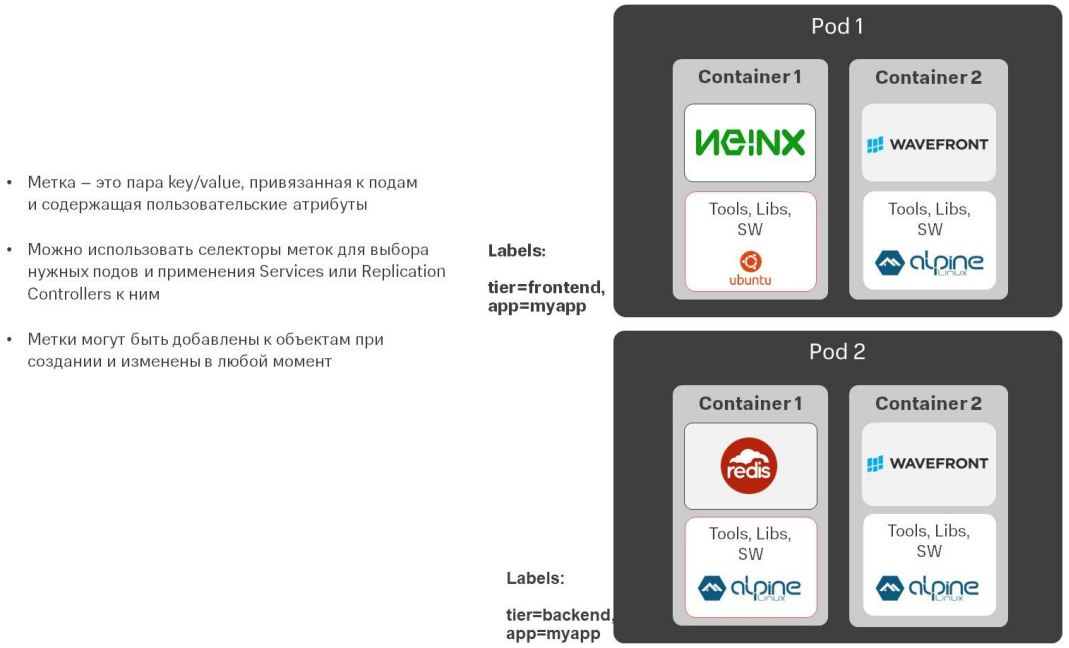
Pods



Под — это минимальная единица, доступная для запуска в среде Kubernetes. Внутри пода может работать сразу несколько контейнеров, объединенных

логически и физически общими ресурсами сети их хранения. Например, распределенные приложения, состоящие сразу из нескольких контейнеров, выполняющих единую логическую задачу.

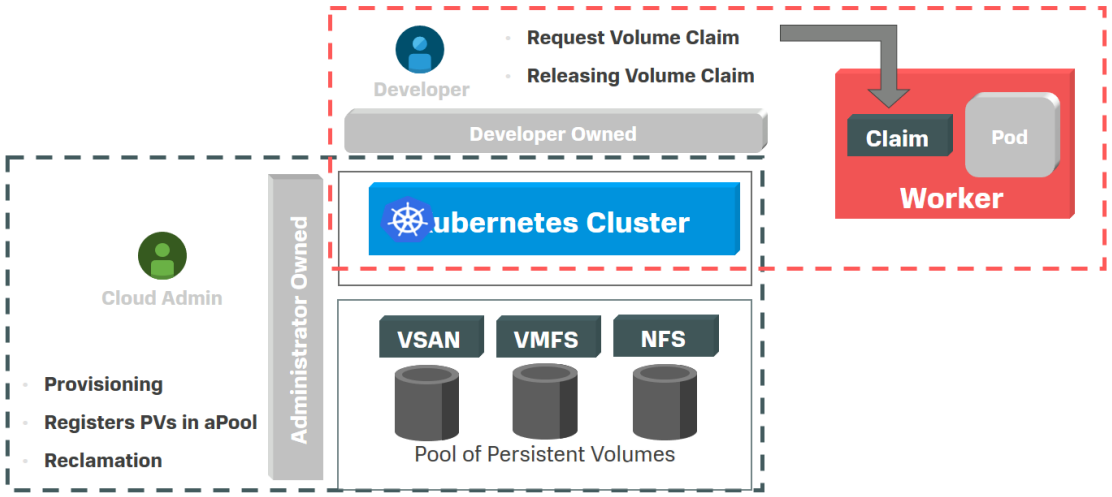
Количество подов в крупных реализациях может достигать сотен и даже тысяч — в этом случае работать с ними напрямую может быть трудозатратно. Намного проще взаимодействовать с ними через тэги.



Persistent Volume Claim

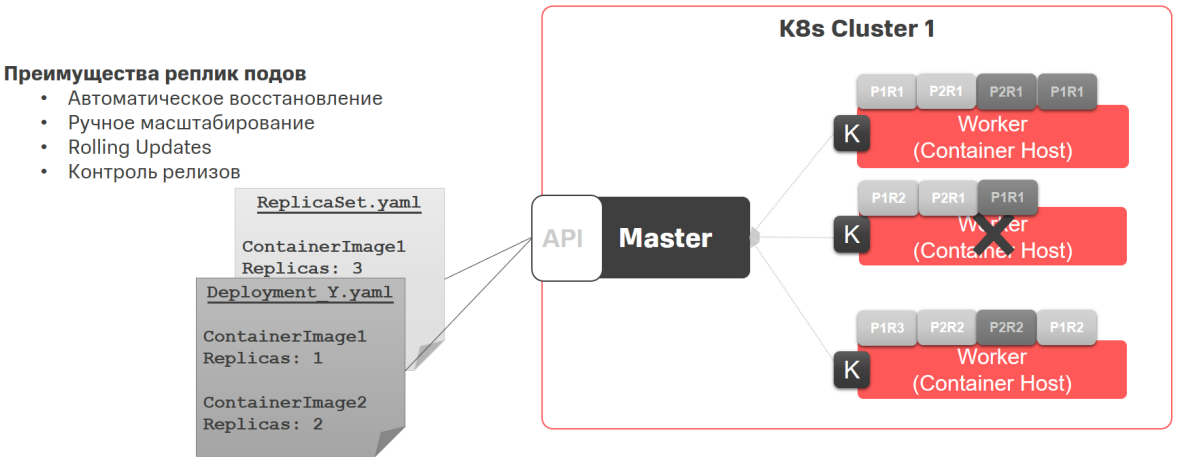
Контейнеры изначально рассматривались как некие эфемерные сущности, которые создаются и удаляются автоматически. Но, как мы знаем, многим приложениям требуется сохранять результаты своей работы. Делать это на неком эфемерном диске, который будет удален при следующем перезапуске, невозможно.

Чтобы решить эту задачу, был внедрен механизм **Persistent Volume**. Он позволяет получить у нижележащей системы хранения данных виртуальный диск для хранения данных внутри пода.



Replication Controller

Служит для контроля за подами в кластере. Он автоматически определяет нужное количество в зависимости от заданных настроек, распределяет их на worker-нодах и автоматически перезапускает поды в случае выхода из строя самих подов или worker-нод.



StatefulSet

Служит для запуска определенного типа приложений, чаще всего многоуровневых, которым требуется определенная очередность запуска. Например, если нужно, чтобы сначала была создана и запущена база данных, затем — некий сервис приложения и только после этого — веб-сервис, который уже будет обслуживать запросы пользователей.

Путь запуска **последовательно** реплик подов.
Позволяет работать подам в clustered mode
- Master/Slave приложения

Важно для приложений, которым нужны:
- Фиксированные и уникальные сетевые идентификаторы
- Фиксированное persistent storage
- Развертывание и масштабирование по запросу

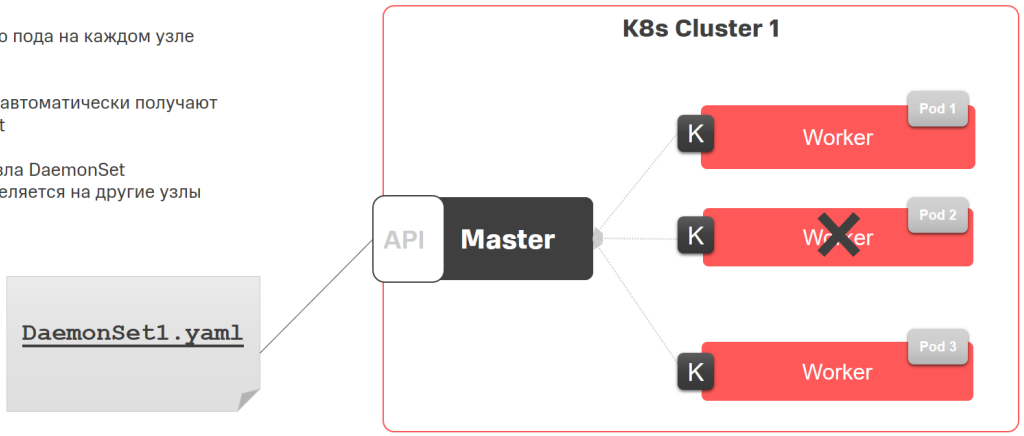
Необходим Headless service

Примеры:
- Zookeeper, Cassandra, etcd, MySQL, etc

DaemonSet

Чаще всего используется для запуска служебных приложений, например для мониторинга и логирования. Позволяет запустить определенные поды на всех нодах кластера.

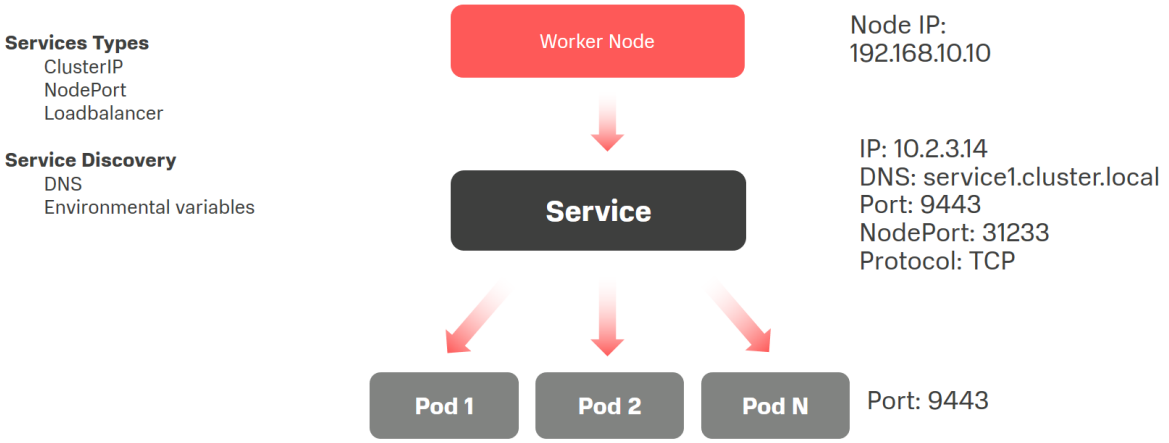
- Запускает копию пода на каждом узле в кластере
- Все новые узлы автоматически получают поды DaemonSet
- При удалении узла DaemonSet не перераспределяется на другие узлы



Service

Поды, как мы знаем, эфемерны, и часто могут менять IP-адрес и некоторые физические значения. Поэтому внешний пользователь может получить доступ к приложению.

Но тут на помощь приходит сущность Service. Она позволяет организовать виртуальный балансировщик нагрузки, который будет следить за текущим количеством подов, их эфемерными адресами и перераспределять нагрузку на них.



Закрепить материал

- **Docker/Kubernetes** https://labs.play-with-k8s.com/
- **Hands-On Labs** https://labs.hol.vmware.com/
- **ModernAppsNinja Courses from VMware** https://modernapps.ninja
- **Learn Kubernetes. From Experts. For Free.** https://kube.academy/
- **K8S** https://kubernetes.io/
- **Cloud Native Apps:** https://blogs.vmware.com/cloudnative